ezVis – An Open-Source, Cross-Platform Visualization Tool

Randall Hand, Paul Adams
Engineer Research and Development Center Major Shared Resource Center
Vicksburg, MS
Randall.E.Hand@erdc.usace.army.mil
Paul.Adams @erdc.usace.army.mil

Abstract

The visualization requirements of today's user community differ significantly from those of even 5 years ago. Terabyte data sets are now produced routinely at the Major Shared Resource Centers (MSRCs), with petabyte-scale data sets on the horizon. Users who formerly exported data sets produced at the MSRC to their local workstation for visualization now find that larger data sets exceed the capacity and capability of their local visualization tools. The majority of users today are physically remote from more powerful visualization technology located at the Scientific Visualization Center (SVC). Although the SVC can deliver preplanned movies in DVD format and some ability exists for users to remotely access SVC resources, mature, well-developed means do not exist for the user to remotely access SVC high performance visualization resources. A potential problem is that investigative visualization will remain difficult for users with terabyte and larger data sets. This is due to several factors including network bandwidth, latency, and infrastructure at the researcher's location. To remedy this problem, the U.S. Army Engineer Research and Development Center (ERDC) MSRC SVC is enhancing the Visualization ToolKit (VTK), an existing open-source library, to deliver remote visualization capability. This enhancement consists of abstracting the calls to the VTK library to make it easier to program for users. This package is called ezVis and is part of the solutionHPC initiative. ezVis has the capability of reading and writing many different file formats, such as Ensight Gold, Plot3D, XDMF, and VTK. ezVis also supports off-screen rendering, which means that no purchase of dedicated graphics hardware is needed. The ezVis package runs on multiple HPC platforms, so the user does not have to migrate files between platforms.

Keywords: Scientific Visualization, Scalable Rendering, Parallel Rendering

1. Introduction

In 2000, the top computer in the world, IBM's ASCI White, had a maximal achieved performance of 2.38 TFLOPS. In 2004, 77 computers were more powerful than ASCI White, which is now 30 times slower than the fastest computer [1]. This trend is expected to continue, with the first PFLOPS machine being available in 2009. With the ubiquitous availability of TFLOPS machines, the associated data are growing at an increasing rate. Terabyte data sets are now common.

Gaining useful information from the data is a challenging task. This task often falls to the researcher or a visualization scientist. Extracting insight from a multiterabyte data set presents the researcher with several problems. These problems include transfer and storage of the data, graphics hardware to visualize it, as well as having visualization software capable of handling the data.

The ERDC MSRC is looking not only at the problem of visualization but also at the whole user experience. solutionHPC [2] is an initiative at the ERDC MSRC to empower the user to solve problems heretofore that were impossible. This is accomplished by creating an interface between the researcher and the machine that is easier to use for the creation and submission of jobs (ezHPC), managing of data (ezStor), and extracting insight from the data (ezVis).

ezViz tackles the visualization problems of the researcher by providing one of two mechanisms. The first, which is currently available, is a batch visualization capability. This batch capability allows the users to create images from their data while it still resides on the supercomputer. These images, which are less than a few megabytes in size, can then be moved with ease to the researcher's workstation. Storage and network bandwidth are no longer a concern when visualizing the data. The second mechanism is to provide a Web interface to visualizing the data. This mechanism is currently under development.

2. ezVis Design Considerations

Several challenges, some of which are outlined above, were the driving force behind the design of ezVis. The first of these challenges is simply the data itself. This includes managing data locality (i.e., the transfer and storage of the data). By leaving the data on the supercomputer, the researcher does not need to worry, at least for ezVis, about moving data from one machine to another. In addition this includes the format of the data itself. Since a standardized format does not exist for writing data from simulations run on the supercomputers, different researchers use different formats. The desire is for ezVis to be able to read in a variety of formats.

The second challenge is to have the necessary software tool to visualize the data. ezVis could be written from "scratch" to meet this design point. However, reinventing the wheel was undesirable. Thus, several commercial off-the-shelf (COTS) products were examined, including Ensight [3] and Visualization Toolkit (VTK) [4], [5]. The difficulty with using a product like Ensight, which supports batch visualization, is that the source code is unavailable. This limits what a researcher can accomplish with the code. In addition, Ensight is not available on all platforms. VTK was chosen because it is an open-source visualization toolkit. Since VTK is open-source, it can be ported to new machines, as they become available, and compiled on them.

Another constraint is that most visualization is performed on machines that have graphics hardware. Most supercomputers, with the exception of some from SGI, do not have graphics hardware available on them. The Mesa three-dimensional (3-D) graphics library [6], henceforth called Mesa, is an open-source graphics library with an application programming interface (API) that is similar to OpenGL. Mesa allows for off-screen rendering by using the central processing unit (CPU). It is not dependant on any graphics hardware. In addition, Mesa support is included in VTK. Mesa implements the latest version of OpenGL, which is currently Version 1.5. Again, since Mesa is open-source, it can be ported to new machines, as they become available, and compiled on them.

The final design point is for ezVis to be simple to code. The argument can easily be made that VTK has all of the ability to meet all the design points, including this one. VTK's strength is that it supports not only the scientific researcher but also many other groups including the medical field. This is also a drawback; how is a researcher not versed in visualization terms to know which VTK functions are appropriate for use? ezVis is a higher level abstraction of VTK that makes the appropriate use of the underlying VTK structure.

3. Usage

ezViz is designed to be used from C and C++ programs through a simple API consisting of three major steps:

1. Creating the ezViz scene
2. Configuring the scene for the desired visualization
3. Calling the required ezViz rendering function

Depending on the desired complexity of the scene, this can be completed in as little as five lines of code. Also, ezViz does not require that individual scenes be constructed for multiple renderings. A single scene can be configured once and passed to several rendering functions to generate multiple outputs or can be reconfigured between rendering functions to generate slightly varying outputs like rotations.

While the intention is for the ezViz code to link directly into the simulation for run-time visualization, it can also be used as a separate process. Writing separate visualization programs of only a few dozen lines of code and compiling them separately enables the execution through various shell or exec functions used by the simulation. This adds a level of error recovery, by separating the visualization from the simulation, such that any errors or crashes in the visualization will not affect the running simulation. This capability also allows for post-processing of data after the simulation has completed.

3.1. Input

ezViz requires that the input data are written to a disk, so that it is in a standard and easily accessible format. Since ezViz is built on top of VTK, it supports all of the formats that VTK supports, and a few more added outside of VTK, as can be seen in Figure 1.

PNG, BMP, JPG, TIF, PNM	Classic Image formats
VTK	VTK Legacy Format
OBJ	WaveFront OBJ Files
PLY	Stanford University Format
POLY	Side Effects PRISM ASCII Format
STL	Common Stereolithography Format
G	Brigham Young University Format
CASE, SOS	Ensight
DEM	Digital Elevation Maps
XYZ	Plot3D
UG	Unigraphics Facet Format
MHA, MHD	UNC MetaImage Format
FACET	Facet Files
CUBE	Gaussian Cube Files
INP	AVS "UCD" Format
VTP, VTU, VTR, VTS, VTI	VTK XML Formats
XDMF	Extensible Data Model Format, by ARL
XMDF	Extensible Data Model Format, by ERDC
PDB	PDB Molecular Data Files

Figure 1. Supported input formats

Various rendering methods require different types of input data. Where possible, data conversion is done automatically; otherwise, ezViz simply displays a warning message and fails to generate the desired output.

3.2. Output

ezViz supports two types of outputs, polygonal models and images.

3.2.1. Polygonal models

Certain rendering techniques (e.g., isosurfacing) generate or extract geometry that is useful for other functions, such as third-party rendering packages. ezViz supports saving the geometry when using these techniques as shown in Figure 2.

OBJ	WaveFront OBJ – Stores Geometry Only
VRML	Virtual Reality Modeling Language - Stores Geometry and Color Information
PLY	Stanford University PLY Format – Stores Geometry and Color Information
STL	Stereolithography Format – Stores Geometry Only

Figure 2. Supported output polygonal formats

In addition to simply storing these output formats, ezViz supports a basic set of postprocessing tools:

- Cleaning – removing duplicate vertices and edges
- Decimation – Quadric Decimate to reduce the size and complexity of the mesh
- Triangulation – converting a polygonal mesh to a triangular mesh (required for some output formats)

Using the polygonal output mode of ezViz, cleaning and decimation of surfaces or models read from other analysis or visualization packages can be done in a batch mode.

3.2.2. Images

All rendering techniques support generation of output images with associated contextual information. Several common formats are supported including PNG, JPG, TIF, BMP, and PNM. These images can be generated at arbitrary sizes, even exceeding OpenGL and Mesa's 4096×4096 limitations, and can be titled and time-stamped. Bounding boxes and color bars can also be enabled for further information. All of these functions are user customizable and can be turned off if desired.

3.3. Visualization techniques

ezViz is capable of visualizing data in several ways. These include rendering a polygonal data set (such as a model), extracting a contour or isosurface, using flow glyphs, or streamlines.

3.2.3. Model rendering

This mode loads a simple polygonal data set from a disk, such as an isosurface or bounding surface generated by another program or another ezViz rendering process. Once loaded, it can simply be rendered to an image or saved as another model to a disk. The latter mode is useful when combined with the Cleaning and Decimation tools to reduce the complexity of models.

3.2.4. Contour extraction

This technique works with volume data sets and extracts isocontours or isosurfaces. The resulting isosurfaces can be colored by any additional field and then saved as an image or a model. The coloring operation can be mapped to an additional scalar or any component or magnitude of a vector field. If the input data set uses cell data instead of point data, then the data will automatically be converted into the appropriate format.

3.2.5. Flow glyphs

In this technique, a volume data set is loaded and one of many simple models, ranging from a three-dimensional arrow to a simple vertex point, is placed at each data point. The resulting complexity can be overwhelming to many systems, so a masking function is in place to allow a statistically random or uniform scattering of points to be used in place of the entire data set. The models can be oriented to a vector field, or left unoriented, and can be scaled by a scalar field in the data set. They can also be colored by any other field or the resulting vector magnitude or scaling factor, and then written to a disk as an image or model.

3.2.6. Streamlines

This technique uses a volume data set and shows streamlines passing through a specified point. More than one streamline can be generated at a time using the spherical or line sources. The resulting lines can be colored by additional fields in the data set or by several parameters resulting from the streamlines calculation (normals, rotation, vorticity, etc.) Several parameters exist to control the types of calculations used for the placement of the streamline and how it will terminate. The result of a streamline rendering can only be saved as an image.

3.2.7. Volume rendering

This technique takes a volume data set and then generates a volume rendering using ray-casting techniques. One is able to map opacity and color to separate values or to the same value. The result can only be saved as an image.

4. Case Studies

4.1.Volume-rendered backhoe

Using a VTK Legacy data set of a synthetic aperture radar image of a backhoe, a volume-rendered image was extracted. The data set is 1024×1024×1024 unsigned chars, and the resulting image is 1280×960. Sixteen processors were used to ray cast the scene in parallel. The relevant code can be seen in Figure 3. The image, which can be seen in Figure 4, took 13.3 minutes to generate on an Onyx 340 with 600-MHz MIPS processors.

The same data set was processed again, but the resulting image was 6400×3600. This requires tiling of the image, as it exceeds the 4K limit in OpenGL. Sixteen processors were used to ray cast in parallel, and 360 images were generated from various angles to create a full rotation. Each frame took an average of 30 minutes, with the entire process taking approximately 6.5 days.

4.2. Isosurface of breaking waves

The 600-time-step Breaking Waves data set is a 512×128×128 grid containing a scalar (Fraction of water in a given cell) and a vector (Velocity) at every point. All 600 frames were processed through ezViz to extract an isosurface from the Fraction field and color map it by Velocity magnitude. All results were written to a disk as 800×600 PNG images. The relevant portion of the code can be see in Figure 5. One of the resulting images can be seen in Figure 6. The ezViz program was wrapped in a shell script to process multiple files simultaneously. Twenty-four frames were processed at a time, each frame taking approximately 15 to 20 seconds. The entire process took approximately 7.5 minutes.

The same data set was processed again, but written as 6400×3600 PNG images. This requires tiling of the image, as it exceeds the 4K limit in OpenGL. The resulting images took ~2:5 minutes each, and the entire process took approximately an hour.

4.3. Polygonal models

The dragon data set consists of 871,414 triangles in Stanford University's PLY format, creating a small Chinese dragon. Using ezViz, this data set was loaded and rendered as an 800×800 PNG image, in 23 seconds. This image can be seen in Figure 7.

The same data set was loaded again and post-processed. It was cleaned and decimated down to 27,076 triangles and saved as a PLY. This took 39 seconds, and the source can be seen in Figure 8. The resulting PLY file was then rendered into a 800×800 PNG image, which took 5.21 seconds and can be seen in Figure 9.

5. Conclusions

ezVis is one portion of a larger program called solutionHPC, to enable high performance computing users to easily generate and understand their data. By basing ezVis on open-source visualization code, cross-platform migration is achieved. HPC users can either integrate ezVis into their own code or run it separately. ezVis is currently undergoing testing with a select group of users at the ERDC MSRC.

6. References

[1]. Top 500 [online] http://www.top500.org .

[2]. Swillie, S. "solutionHPC on the Way," *Resource*, ERDC MSRC, Fall 2004.

[3]. Computational Engineering International, Inc., "Ensight User Manual for Version 8.0," North Carolina, USA, 2005.

[4]. Shroder, B., Martin, K., Lorenson, B. *The Visualization Toolkit, An Object-Oriented Approach To 3D Graphics, 3rd edition.* 2003.

[5]. Kitware *The Visualization Toolkit User's Guide.* 2001.

[6]. Mesa, [online] http://www.mesa3d.org .

```
// 1-time setup
scene = ezVizInitialize();
strcpy(scene->input_filename, argv[1]);
scene->output_height = 3600;
scene->output_width = 6400;
scene->showTitle = 0;
scene->showTimeStamp = 0;

scene->enableColorMap = 1;
scene->showColorBar = 1;
sprintf(scene->colorField, "volume_scalars");
scene->colorBarOrientation = ezViz_OrientHorizontal;
scene->colorBarLocation = ezViz_SideBottom;
scene->colorFieldComponent = ezViz_RangeX;
scene->boundingBox = 0;

scene->Elevation = -75;
scene->Pitch = 1;
scene->Zoom = 5;

// Build the colormap, since the default uses full opacity at all values
ezViz_AddColorMapEntryRGB(scene,  0.0, 0,0,0,0.00);
ezViz_AddColorMapEntryRGB(scene, 36.5, 0,0,1,0.16);
ezViz_AddColorMapEntryRGB(scene, 73.0, 0,1,1,0.33);
ezViz_AddColorMapEntryRGB(scene,109.5, 0,1,0,0.50);
ezViz_AddColorMapEntryRGB(scene,146.0, 1,1,0,0.66);
ezViz_AddColorMapEntryRGB(scene,182.5, 1,0,0,0.83);
ezViz_AddColorMapEntryRGB(scene,219.0, 1,0,1,1.00);
ezViz_AddColorMapEntryRGB(scene,255.0, 1,1,1,1.00);

ezVizSetNumberOfThreads(24);
for(scene->Roll = 0; scene->Roll < 360; scene->Roll++) {
      sprintf(scene->output_filename, "frame%04i.png", scene->Roll);
      ezVizRenderVolume(scene);
}
```

Figure 3. Source for the backhoe volume rendering

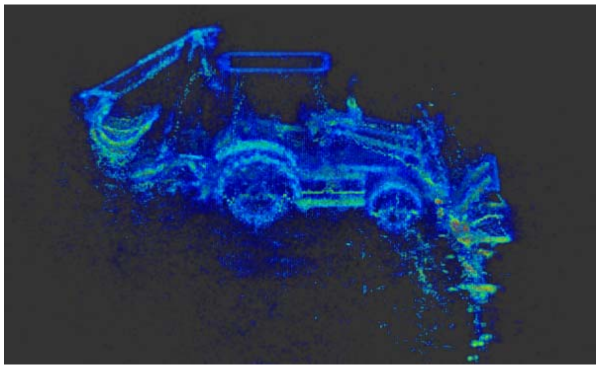
Figure 4. Small backhoe volume rendering

```
// 1-time setup
scene = ezVizInitialize();
scene->output_height = 600;
scene->output_width = 800;
scene->contourIsoValue = 0.5;
sprintf(scene->contourFieldName, "Fraction");

scene->enableColorMap = 1;
sprintf(scene->colorField, "Velocity");
scene->colorBarOrientation = ezViz_OrientHorizontal;
scene->colorBarLocation = ezViz_SideBottom;
scene->colorFieldComponent = ezViz_RangeVectorMagnitude;
scene->show_Statistics = 0;

scene->autoStretchColorMap = 0;
ezViz_AddColorMapEntryHSV(scene, 0, 0.6667, 1, 1);
ezViz_AddColorMapEntryHSV(scene, 1.5, 0.3333, 1, 1);
ezViz_AddColorMapEntryHSV(scene, 3, 0, 1, 1);
scene->Zoom = 1.5;

// every time setup
for(i=1; i<argc; i++) {
    strcpy(scene->input_filename, argv[i]);
    sprintf(scene->output_filename, "%s-iso.png", argv[i]);
    ezVizRenderContour(scene);
}
```

Figure 5. Source for the breaking waves isosurface

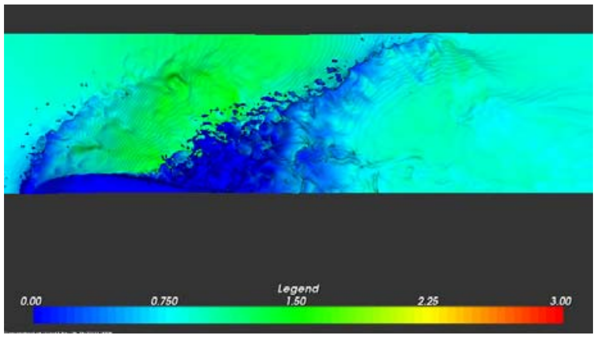

Figure 6. Small breaking waves frame

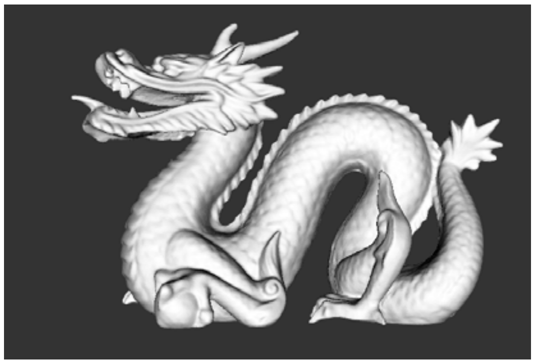
Figure 7. Dragon

```
// 1-time setup
scene = ezVizInitialize();
scene->input_format = ezVizFile_AUTO;
scene->output_format = ezVizOutput_PLY;
scene->show_Statistics = 1;

scene->enableColorMap= 0;
scene->modelDecimate = 1;
scene->modelClean = 1;
scene->decimate_Subdivisions = 50;

// every time setup
for(i=1; i<argc; i++) {
    strcpy(scene->input_filename, argv[i]);
    sprintf(scene->output_filename, "%s.png",  rgv[i]);
    strcpy(scene->title, argv[i]);
    ezVizRenderModel(scene);
}
```

Figure 8. Source for the decimated dragon

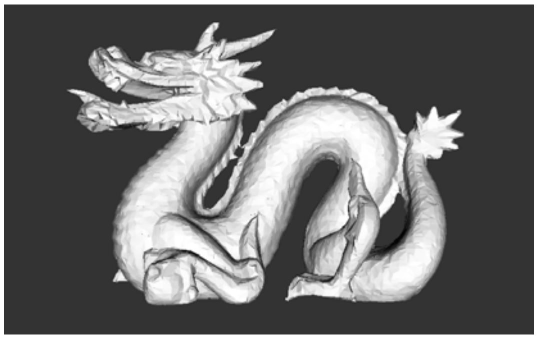
Figure 9. Dragon, decimated